# Technical Breakdown of Position Based Fluid

Qingyuan Qie
will.qie@mail.utoronto.ca
University of Toronto
Toronto, ON, Canada

Changlin Su
sheldon.su@mail.utoronto.ca
University of Toronto
Toronto, ON, Canada

## ABSTRACT

This paper provides a technical breakdown of implementing a parallel version of the Position Based Fluids(PBF)[5] method. We discuss the simulation algorithms of PBF, the effects of different confinements in the algorithms. Due to the highly parallelizable nature of the algorithm, we provide an implementation using the CUDA platform which achieves realtime simulation with a particle count of 27k. A CPU version is also available.

## CCS CONCEPTS

• **Computing methodologies → Physical simulation**.

## KEYWORDS

Fluids Simulation, Physics-Based Animation

**ACM Reference Format:**
Qingyuan Qie and Changlin Su. 2021. Technical Breakdown of Position Based Fluid. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/0.0/0.0

## 1 INTRODUCTION

In Position Based Fluids(PBF) method, fluids are simulated with discrete particles, the surfaces of the fluids are reconstructed at render time based on the position of the underlying particles. In each time step, the PBF method first predicts the positions and velocity. Then it corrects the particle positions by enforcing the incompressibility constraints. The new velocity of the particles are computed from the new positions and the original positions of the particles.

Algorithm 1 is an overview of a simulation step in PBF method.

## 2 ENFORCING INCOMPRESSIBILITY

For particle $i$ at position $p_i$, we compute the density of the fluid around particle $i$ using the estimator:

$$\rho_{F(i)} = \sum_{j \in F(i)} m_j W_{poly6}(\boldsymbol{p}_i - \boldsymbol{p}_j, h) \tag{1}$$

where $\rho_0$ is the rest density of the fluid, $m_j$ is the mass of the particle $j$, $h$ is a constant, and $W$ is the Poly6 kernel from [6]. $F$ is the neighbor function that returns the neighboring particle of particle $i$.


*Conference'17, July 2017, Washington, DC, USA*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/0.0/0.0

**Algorithm 1** simulation step

**for** all particles $i$ **do**: ▷ Fluid advect
  apply forces $\boldsymbol{v}_i = \boldsymbol{v}_i + \Delta t f_{ext}$
  predict position $\boldsymbol{x}_i^* = \boldsymbol{x}_i + \Delta t \boldsymbol{v}_i$
**end for**
**for** all particles $i$ **do**:
  find neighboring particles $F(\boldsymbol{x}_i^*)$
**end for**
**while** $iter < solverIterations$ **do**: ▷ Iterativly solves imcompressibility constraints
  **for** all particles $i$ **do**:
    calculate $\lambda_i$
  **end for**
  **for** all particles $i$ **do**:
    calculate $\Delta \boldsymbol{p}_i$
    perform collision detection and response
  **end for**
  **for** all particles $i$ **do**:
    update position $\boldsymbol{x}_i^* = \boldsymbol{x}_i^* + \Delta \boldsymbol{p}_i$
  **end for**
**end while**
**for** all particles $i$ **do**:
  update velocity $\boldsymbol{v}_i = \frac{1}{\Delta t}(\boldsymbol{x}_i^* - \boldsymbol{x}_i)$
  apply viscosity
  update position $\boldsymbol{x}_i = \boldsymbol{x}_i^*$
**end for**

The Poly6 kernel is defined as follows,

$$W_{poly6}(\boldsymbol{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3 & 0 \le r \le h \\ 0 & \text{otherwise} \end{cases}$$

where $r$ is the norm of $\boldsymbol{r}$. Combine equation (1) and the definition of Poly6, we can notice that if the distance between particle $i$ and $j$ is greater than $h$, particle $j$ does not contribute to the density around particle $i$.

Then we introduce the constant density constraints $C_i$. For particle $i$, we have,

$$C_i(\boldsymbol{p}) = \frac{\rho_i}{\rho_0} - 1 \tag{2}$$

where $\rho_0$ is a constant that denotes the rest density of the fluid, and $\boldsymbol{p}$ is the position of all the particles in the system.

Since we want the density around each particle is always equal to $\rho 0$, we want to find a position correction $\Delta \boldsymbol{p}$ such that,

$$C(\boldsymbol{p} + \Delta \boldsymbol{p}) = 0 \tag{3}$$

The solution to (3) is found by a series of Newton steps along the constraint gradient,

$$\Delta(\boldsymbol{p}) \approx \nabla C(\boldsymbol{p}) \lambda \tag{4}$$

$$C(\boldsymbol{p} + \Delta\boldsymbol{p}) \approx C(\boldsymbol{p}) + \nabla C^T \Delta\boldsymbol{p} = 0 \tag{5}$$

$$\approx C(p) + \nabla C^T \nabla C\lambda = 0 \tag{6}$$

And the gradient of the constraint $C_i$ with respect to a particle $k$ is given by,

$$\nabla_{\boldsymbol{p}_k} C_i = \frac{1}{\rho_0} \sum_{j \in F(i)} \nabla_{\boldsymbol{k}} W(\boldsymbol{p}_i - \boldsymbol{p}_j, h) \tag{7}$$

Which has two different cases based on whether $k$ is a neighboring particle or $k$ is the particle $i$ itself,

$$\nabla_{\boldsymbol{p}_k} C_i = \frac{1}{\rho_0} \begin{cases} \sum_j \nabla_{\boldsymbol{p}_k} W(\boldsymbol{p}_i - \boldsymbol{p}_j, h) & \text{if } k = i \\ -\nabla_{\boldsymbol{p}_k} W(\boldsymbol{p}_i - \boldsymbol{p}_j, h) & \text{otherwise} \end{cases} \tag{8}$$

The original paper uses the Spiky kernel fot the gradient computation, which is defined as,

$$\nabla W_{spiky}(\boldsymbol{r}, h) = -\frac{45}{64\pi h^6} \begin{cases} (h - r)^2 \hat{\boldsymbol{r}} & 0 \le r \le h \\ 0 & \text{otherwise} \end{cases}$$

where $\hat{r}$ is the normalized $\boldsymbol{r}$.

Plug (8) into (6) and solves for $\lambda$ yields,

$$\lambda_i = -\frac{C_i(\boldsymbol{p})}{\sum_k |\nabla_{\boldsymbol{p}_k} C_i|^2} \tag{9}$$

Then the position correction $\Delta\boldsymbol{p}_i$ including affect from neighboring particles is

$$\Delta\boldsymbol{p}_i = \frac{1}{\rho_0} \sum_j (\lambda_i + \lambda_j) \nabla W(\boldsymbol{p}_i - \boldsymbol{p}_j, h) \tag{12}$$

And we have the position update formula,

$$\boldsymbol{p}_i^* = \boldsymbol{p}_i + \Delta\boldsymbol{p}_i \tag{13}$$

## 3 TENSILE INSTABILITY

The original PBF paper adds an artificial pressure to mitigate the issue of particle clamping. It introduced an additional correction coefficient defined as,

$$s_{corr} = -k \left( \frac{W(\boldsymbol{p}_i - \boldsymbol{p}_j, h)}{W(\Delta\boldsymbol{q}, h)} \right)^n \tag{14}$$

where $\Delta\boldsymbol{q}$ is a point some fixed distance inside the smoothing kernel radius and $k$ is a small positive constant. In our implementation, we take $k = 0.1$, $|\Delta\boldsymbol{q}| = 0.1h$ and $n = 4$. Then the modified position update formula becomes,

$$\Delta\boldsymbol{p}_i = \frac{1}{\rho_0} \sum_j (\lambda_i + \lambda_j + s_{corr}) \nabla W(\boldsymbol{p}_i - \boldsymbol{p}_j, h) \tag{15}$$

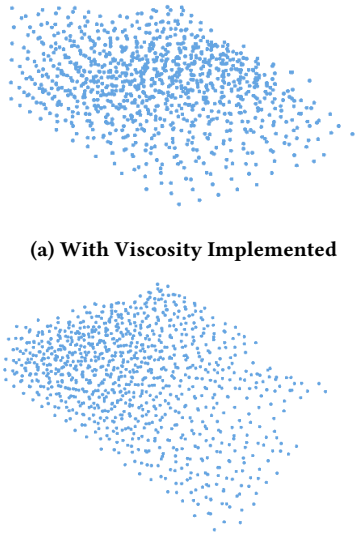
## 4 VORTICITY CONFINEMENT AND VISCOSITY

The original PBF paper implemented vorticity to replace lost energy. However, due to the time constraint, we did not implement vorticity confinement.

We implemented viscosity, which is defined as follows,

$$\boldsymbol{v}_i^* = \boldsymbol{v}_i + c \sum_j \boldsymbol{v}_{ij} \cdot W(\boldsymbol{p}_i - \boldsymbol{p}_j, h). \tag{16}$$

where $\boldsymbol{v}_{ij} = \boldsymbol{v}_j - \boldsymbol{v}_i$. The parameter $c$ is chosen to be 0.001 in our simulation after a few trail and error. The viscosity is important for convincing fluid motion. Figure 1 shows the effect of the viscosity.



**(a) With Viscosity Implemented**

**(b) Without Viscosity Implemented**

**Figure 1: Comparison of the fluid motion with/without viscosity implemented, taken at the same time step.**

When viscosity is not implemented, the particles are more scattered and have the tendency of not moving together, which is not realistic. With viscosity implemented, the fluid is more "thick" visually and thus more similar to how real water moves.

## 5 BOUNDARY AND COLLISION RESPONSE

In our implementation, the fluid is confined in a cuboid whose surfaces are parallel to the coordinate planes. We deploy a simple collision scheme as describe in [7]. Particles that go out of bound in the fluid advection is clamped into the cuboid with an small distance to the boundary, and velocity of the particle is set to 0.

## 6 GRID-BASED NEIGHBOR FINDING

The Poly6 and Spiky function both return 0 when the distance of two particles is greater than $h$. Therefore, we adapted the grid-based neighbor finding algorithm described in [3]. We implemented the CPU version ourselves, and we modified the source code from [2] for CUDA version.

## 7 IMPLEMENTATION DETAILS

Table 1 shows the correspondence of algorithm steps to the name of the functions in our code. Since Algorithm 1 is highly parallelizable with the update of each particle can be computed in parallel, we launch one thread for each particle in the CUDA implementation.

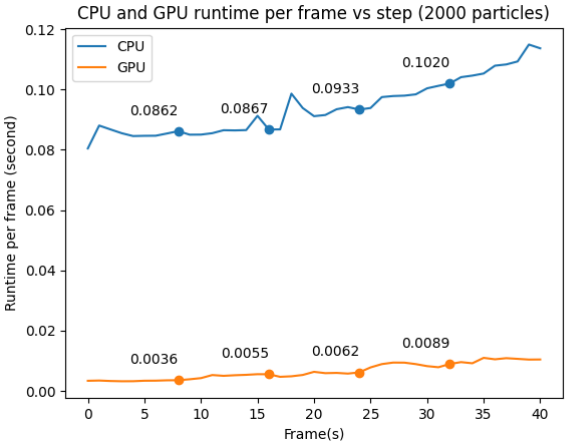



**Figure 2: Performance Comparison between CPU ans GPU, 2k particles**

| Line Number | CUDA Kernel Name CPU Function Name |
|---|---|
| 1 - 4, 14 | advect |
| 5 - 7 | build_grid |
| 9 - 11 | compute_lambda |
| 12-15 | compute_delta_position |
| 16-19 | update_position |
| 21 | update_velocity |
| 22 | viscosity_confinement |

**Table 1: Actual Function name to the pseudo-code in Algorithm 1**

## 8 RESULT

The results were collected on a PC equipped with a Rayen 3700x CPU and RTX3070Ti GPU.

Figure 2 shows the time taken to generate one frame for both the CPU and GPU implementation. We can see that the GPU implementation is approximately 25x faster than the CPU implementation.

Increase number of particles, the CPU implementation becomes painfully slow. We show the results of the GPU implementation at particle count of 12k, 27k and 125k in table 2.

| Particle Count | Time per Frame | Frame per Second(fps) |
|---|---|---|
| 2k | 0.006s | 166fps |
| 12k | 0.04s | 25fps |
| 27k | 0.080s | 12fps |

**Table 2: Performance of GPU implementation**

We can see that we achieved real time simulation performance on all of the three particle count performance in table 2.

Figure 3 is a screenshot from a simulation with 12k particles. The fluid was dropped 2 meters above the floor. We can see from the figure that the fluid have bounded back up from hitting the corner.

**Figure 3: Screenshot from simulation with 27k particles**

## 9 DISCUSSION

There are two major drawbacks of this implementation.

Firstly, the simulation system runs on GPU and it copies the updated particle positions back to CPU for `libigl` [4] to render, which incurs memory transfer costs. This overhead can be completely removed by using an OpenGL render pipeline and leveraging the CUDA-OpenGL Interoperability[1].

Another drawback is that the surface of the fluid was not reconstructed. Further work is required to render the surface of the fluid, which is outside the scope of this project.

Lastly, the GPU implementation was not well-optimized due to time constraint. Further optimization will help achieve a better performance.

## 10 CONCLUSION

In this paper, we implemented the Position Based Fluids simulation algorithm on both CPU and GPU. We achieved real time simulation performance on the GPU implementation.

## REFERENCES

[1] Nvidia Corp. 2021. *CUDA OpenGL Interoperability.* https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__OPENGL.html
[2] Nvidia Corp. 2021. *Cuda Samples, Particles.* https://github.com/NVIDIA/cuda-samples/tree/master/Samples/particles
[3] Simon Green. 2010. *Particle Simulation using CUDA.* https://developer.download.nvidia.com/assets/cuda/files/particles.pdf
[4] Alec Jacobson. 2021. *libigl.* https://github.com/libigl/libigl/
[5] Miles Macklin and Matthias Müller. 2013. Position Based Fluids. *ACM Trans. Graph.* 32, 4, Article 104 (jul 2013), 12 pages. https://doi.org/10.1145/2461912.2461984
[6] Matthias Müller, David Charypar, and Markus Gross. 2003. Particle-Based Fluid Simulation for Interactive Applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (San Diego, California) *(SCA '03)*. Eurographics Association, Goslar, DEU, 154–159.
[7] Keren Zhu. 1999. *The Simulation, Rendering and Application of Incompressible Fluids.* https://github.com/naeioi/PBF-CUDA/blob/master/paper/simu-fluids-keren.pdf